**DN 4.0 JAVA FSE SOLUTIONS – WEEK 3**

**SKILL: Spring Data JPA with Spring Boot, Hibernate**

**Spring Data JPA - Quick Example**

**Creating Database Table**

CREATE DATABASE ormlearn;

USE ormlearn;

**Create Table:**

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

**Insert Sample Records:**

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

**src/main/resources/application.properties**

# Logging  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=harshini21  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**com.cognizant.ormlearn.model**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Country{" +

"code='" + code + '\'' +

", name='" + name + '\'' +

'}';

}

}

**com.cognizant.ormlearn.repository**

package com.cognizant.ormlearn.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**com.cognizant.ormlearn.service**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**src/main/java/com/cognizant/ormlearn/OrmLearnApplication.java**

package com.cognizant.ormlearn;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static CountryService countryService;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

LOGGER.info("Inside main");

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

List<Country> countries = countryService.getAllCountries();

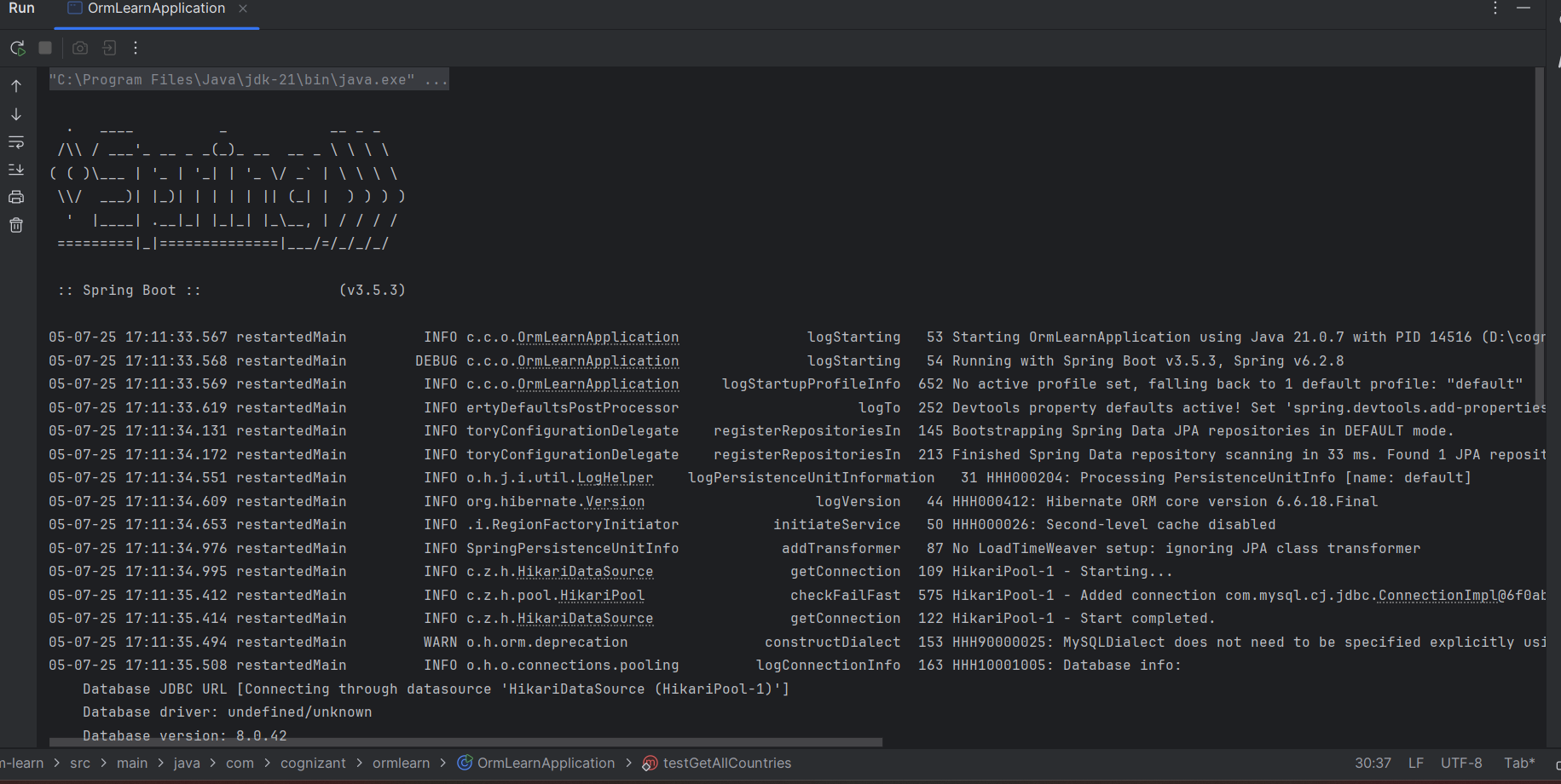
LOGGER.debug("countries={}", countries);

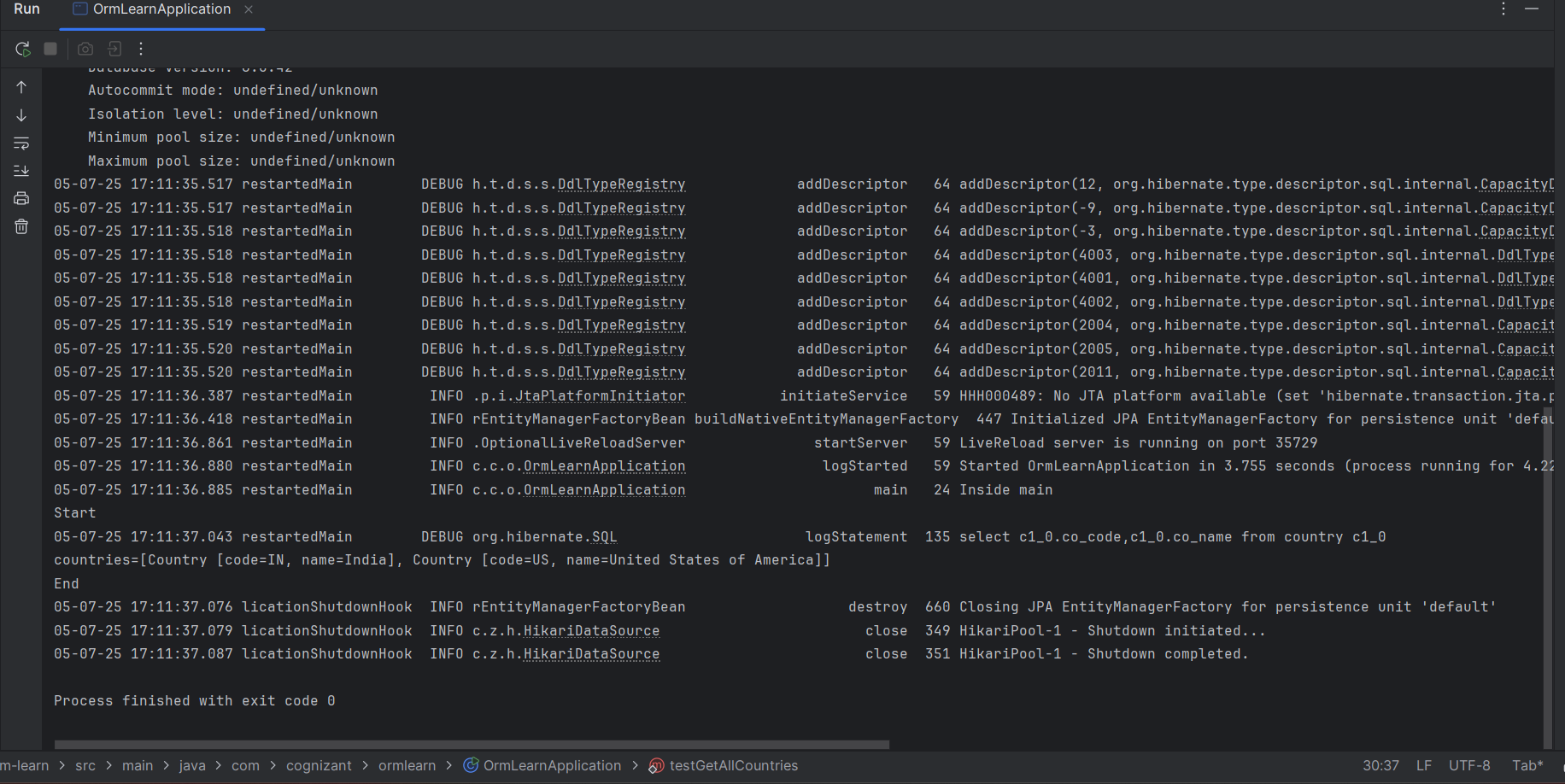
LOGGER.info("End");

}

}

**Output**





**Difference between JPA, Hibernate and Spring Data JPA**

**Java Persistence API (JPA) -The Specification**

* JPA is a specification (JSR 338) for persisting, reading, and managing Java objects in a relational database.
* It is not an implementation itself; rather, it provides a standard set of interfaces and annotations.
* Hibernate is a popular implementation of JPA

**Hibernate -The Implementation**

* Hibernate is an object-relational mapping solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.
* Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.
* It manages the database operations like session management, transaction control, and query execution.
* Hibernate can also be used directly without JPA using its own APIs.

**Spring Data JPA- The Abstraction Layer**

* Spring Data JPA is not a JPA implementation but a higher-level abstraction built on top of JPA (usually with Hibernate).
* It reduces boilerplate code significantly by providing automatic implementations of repositories.
* It integrates with Spring’s transaction management and simplifies common data access operations.

**Code Comparison**

**Using Hibernate (Without Spring)**

// Method to CREATE an employee in the database

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Using Spring Data JPA**

EmployeeRepository.java

java

Copy code

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

java

Copy code

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Summary**

|  |  |
| --- | --- |
| **Approach** | **Criteria** |
| JPA | Standardization |
| Hibernate | Full control/customization |
| Spring Data JPA | Ease of use/productivity |

**For most modern Spring Boot applications, Spring Data JPA is the best choice** because it saves time, reduces boilerplate, and integrates seamlessly with Spring features.